Types & Programming Languages
Exercises 1 (Partial Solutions)

These exercises are based on the material in Lectures 1 and 2.

1. For each of the following expressions, show its sequence of reductions as far as possible, and give a full derivation of each reduction step.

   (a)

$$\begin{array}{c}\mathsf{(if\ 1{=}{=}1\ then\ 2\ else\ 3){+}(1{+}2)}\\ \downarrow\\ \mathsf{(if\ true\ then\ 2\ else\ 3){+}(1{+}2)}\\ \downarrow\\ \mathsf{2{+}(1{+}2)}\\ \downarrow\\ \mathsf{2{+}3}\\ \downarrow\\ \mathsf{5}\end{array}$$

   Full derivation of the first reduction:

$$\dfrac{\dfrac{\dfrac{}{\mathsf{1{=}{=}1} \to \mathsf{true}}\ \text{R-EqT}}{\mathsf{if\ 1{=}{=}1\ then\ 2\ else\ 3} \to \mathsf{if\ true\ then\ 2\ else\ 3}}\ \text{R-IfC}}{\mathsf{(if\ 1{=}{=}1\ then\ 2\ else\ 3){+}(1{+}2)} \to \mathsf{(if\ true\ then\ 2\ else\ 3){+}(1{+}2)}}\ \text{R-SumL}$$

   (b)

$$\begin{array}{c}\mathsf{if\ 1{+}2\ then\ 3\ else\ 4}\\ \downarrow\\ \mathsf{if\ 3\ then\ 3\ else\ 4}\\ \textit{stuck}\end{array}$$

2. Add a new form of expression: $e * e$.

   The reduction rules are almost identical to those for addition.

   u * v $\to$ w if u and v are integer literals and w is their product.

$$\dfrac{\mathsf{e} \to \mathsf{e'}}{\mathsf{e * f} \to \mathsf{e' * f}} \qquad \dfrac{\mathsf{e} \to \mathsf{e'}}{\mathsf{v * f} \to \mathsf{v * e'}}$$

3. The shortcut reduction rules for logical or are as follows:

   true or e $\to$ true

   false or e $\to$ e

4. (a) The relationship between complete execution on the stack and complete reduction should be as follows; it would be possible to prove these statements (exercise!).

   If $e \to^* v$ then executing $\mathit{compile}(e)$ with an initially empty stack results in a stack containing the value v.

   If $e \to^* e'$ and e' is stuck and not a value, then executing $\mathit{compile}(e)$ results in a run-time error at some point.

   The converse of these statements should also be true.

For individual reduction steps and individual stack instructions, we can make the following (rather vague) statements.

Each add instruction corresponds to a reduction $\mathsf{u} + \mathsf{v} \to \mathsf{w}$, and similarly for other operations.

Reductions such as $e + f \to e' + f$ or $v + e \to v + e'$ correspond to executing instructions within a sequence of code.

More precise statements could be formulated and proved (exercise!).

(b) The difference between stack execution and reduction of conditional expressions is that (with the current definitions) the stack machine evaluates both branches of a conditional before evaluating the condition.

To modify the reduction rules so that they match the stack machine, we can treat conditional expressions in the same way as operators such as +, but with 3 operands. We can also define the rules so that we have the same order of evaluation. Here are suitable rules:

$\mathsf{if\ true\ then\ u\ else\ v} \to \mathsf{u}$ if u and v are values.

$\mathsf{if\ false\ then\ u\ else\ v} \to \mathsf{v}$ if u and v are values.

$$\frac{\mathsf{e} \to \mathsf{e'}}{\mathsf{if\ f\ then\ f'\ else\ e} \to \mathsf{if\ f\ then\ f'\ else\ e'}}$$

$$\frac{\mathsf{e} \to \mathsf{e'}}{\mathsf{if\ f\ then\ e\ else\ v} \to \mathsf{if\ f\ then\ e'\ else\ v}}$$

$$\frac{\mathsf{e} \to \mathsf{e'}}{\mathsf{if\ e\ then\ v'\ else\ v} \to \mathsf{if\ e\ then\ v'\ else\ v}}$$

But of course we don't want conditional expressions to be evaluated in this way; in a programming language that can express non-termination, it is important not to evaluate both branches of a conditional. The stack machine needs to be extended with conditional branching, so that we can adapt it to match our original reduction rules for conditionals (exercise: work out the details).